1. In OpenMP, wenn mehrere Threads gleichzeitig ausgeführt werden, kann das Betriebssystem eine Lastausgleich durchführen, das heißt, es kann Threads von einem logischen Kern zu einem anderen verschieben, um eine gleichmäßige Auslastung aller logischen Kerne zu versuchen. Wenn also die Ausführungsreihenfolge der Threads im Programm nicht explizit angegeben ist, kann die Ausführungsreihenfolge der Threads zufällig sein. Das ist der Grund, warum die Threads aus Listing 3.1 ihre Nummern zufällig anzeigen. In Listing 3.2 wurde möglicherweise eine Art Synchronisationsmechanismus (wie Barriers oder ordered Konstrukte) verwendet, um sicherzustellen, dass die Threads in einer bestimmten Reihenfolge ausgeführt werden.

2.Über 3.10:
Der Code verwendet die OpenMP-Direktive #pragma omp parallel for collapse(2), um verschachtelte Schleifen zu parallelisieren. Die collapse(2)-Klausel kombiniert zwei Schleifen in eine einzige große Schleife, die Iterationen werden dann auf verschiedene Threads verteilt. Dies steigert die Parallelität, da mehr Iterationen gleichzeitig ausgeführt werden können.

Jede Iteration berechnet die Funktion 'neighbors' für ein Element des Arrays 'plane' und aktualisiert das Array 'aux_plane' basierend auf der Anzahl der Nachbarn. Solche unabhängigen Iterationen eignen sich gut für Parallelisierung.

Die Effizienz hängt jedoch von der Größe der Variable 'size' ab. Bei großem 'size' kann die Parallelisierung die Leistung deutlich verbessern, da Berechnungen auf mehreren Kernen gleichzeitig durchgeführt werden.

Die Komplexität der Funktion 'neighbors' ist ebenfalls zu berücksichtigen. Bei rechenintensiver Funktion ist Parallelisierung vorteilhafter, um die Rechenkapazität von Multicore-Systemen zu nutzen.

Insgesamt scheint der Code Multicore effektiv zu nutzen, indem er verschachtelte Schleifen parallelisiert und mehrere logische Kerne einsetzt.

Performance Impact:

Das Ersetzen des Standard-Zufallszahlengenerators durch die Funktion ‚rnd‘ kann die Qualität der Zufälligkeit beeinflussen, könnte aber aufgrund des geringeren Rechenaufwands von ‚rnd‘ die Leistung verbessern.

Unabhängige Seeds für jeden Thread beseitigen Abhängigkeiten zwischen den Threads und können so die Effizienz der Parallelisierung erhöhen.